Parallel Algorithms for Collective Processes in High Intensity Rings

Andrei Shishlo, Jeff Holmes, and Viatcheslav Danilov
Oak Ridge National Laboratory, SNS Project, 701 Scarboro Rd, MS-6473, Oak Ridge TN, USA 37830
{shishlo, vux, jzh}@sns.gov

Abstract. Computational three-dimensional space charge (3DSC) and wake field force algorithms were developed and implemented into the ORBIT computer code to simulate the dynamics of present and planned high intensity rings, such as PSR, Fermilab Booster, AGS Booster, Spallation Neutron Source (SNS), and proton driver. To provide affordable simulation times, the 3DSC algorithm developed for ORBIT has been parallelized and implemented as a separate module into the UAL 1.0 library, which supports a parallel environment based on MPI. The details of these algorithms and their parallel implementation are presented, and results demonstrating the scaling with problem size and number of processors are discussed.

1 Introduction

Collective beam dynamics will play a major role in determining losses in high intensity rings. The details of these processes are so complicated that a good understanding of the underlying physics will require careful computer modeling. In order to study the dynamics of high intensity rings, a task essential to the SNS project [1], we have developed direct space charge and impedance models in the macro-particle tracking computer code, ORBIT [2,3]. Initially, separate transverse space charge and longitudinal space charge/impedance models were developed, benchmarked, and applied to a number of problems [4,5]. We have now extended the impedance model to include the calculation of forces due to transverse impedances and, because such forces depend on the longitudinal variation of the beam dipole moments, the space charge model has been extended to three dimensions. In many cases, the resulting simulations including 3DSC calculations will require tracking tens of millions of interacting macro-particles for thousands of turns, which constitutes a legitimate high performance computing problem. There is little hope of carrying out such calculations in a single processor environment. In order to meet the need for credible simulations of collective processes in high intensity rings, we have developed and implemented the parallel algorithms for the calculation of these processes[1].

1.1 Parallel algorithms

The main goals of parallel computer simulations are to shorten the tracking time and to provide for the treatment of larger problems. There are two possible situations for

[1] Research on the Spallation Neutron Source is managed by UT-Battelle, LLC, under contract DE-AC05-00OR22725 for the U.S. Department of Energy.

tracking large numbers of particles with macro-particle tracking codes such as ORBIT. In the first case, particles are propagated through the accelerator structure independently without taking into account direct or indirect interactions among them, so there is no necessity for parallel programming. It is possible to run independent calculations using the same program with different macro-particles on different CPUs and to carry out the post-processing data analysis independently. In the opposite case there are collective processes, and we must provide communication between the CPUs where programs are running. Unfortunately, there is no universal efficient parallel algorithm that can provide communication for every type of collective processes. The best parallel flow logic will be defined by the mathematical approach describing the particular process and the ratio between computational and communication bandwidth. Therefore, our solutions for parallel algorithms cannot be optimal for every computational system.

Our implementation of parallel algorithms utilizes the Message-Passing Interface (MPI) library. The timing analysis has been carried out on the SNS Linux workstation cluster including six dual i586 CPUs with each having 512 kBytes L2 cache, 512 MB RAM and the 100 Mb/s Fast Ethernet switch for communication. The communication library MPICH version 1.2.1, a portable implementation of MPI, has been installed under the Red Hat 7.0 Linux operating system.

2 Transverse Impedance Model

The transverse impedance model in ORBIT [3] is based on an approach previously implemented in the longitudinal impedance model [5], which calculates the longitudinal kick by summing products of Fourier coefficients of the current with the corresponding impedance values, all taken at harmonics of the ring frequency [6]. A complication with the transverse impedance model arises due to the betatron motion, which has much higher frequency than synchrotron motion. Consequently, the harmonics of the dipole current must include the betatron sidebands of the revolution harmonics. Because of this and the fact that the number of transverse dimensions is two, the transverse impedance model requires four times as many arrays and calculations as does the longitudinal impedance. In the transverse impedance model, the kicks are taken to be delta-functions. This approximation is valid when the betatron phase advance over the physical extent of the impedance is small. If this is not the case, the impedance must be represented as a number of short elements, which is valid when the communication between elements is negligible. One exception to this rule is the resistive wall impedance. Because the resistive wake does not involve propagating waves, it can be treated as localized away from synchrobetatron resonances. When communication between elements is significant, then a more general Green's function approach, which is beyond the scope of this model, must be used.

2.1 Parallel Algorithm for Transverse Impedance Model

The parallel algorithm for ORBIT's transverse impedance model has been developed assuming that propagated macro-particles are arbitrarily distributed among CPUs. Typically, we must consider only a few transverse impedance elements in the accelerator lattice, and the resulting calculation time is small. Consequently, we derive this algorithm more for simplicity than for efficiency.

The parallel flow logic for the transverse impedance model is shown in Table 1. There are only two stages of calculation in which data is exchanged between CPUs. In the first stage the maximal and minimal longitudinal coordinates of all macro-particles must be determined. We used the MPI_Allreduce Collective Communication MPI function with the MPI_MAX parameter describing MPI-operation to find maximal and minimal values of the longitudinal coordinates for all CPUs. In the 5-th step we sum the array of transverse kick values for all CPUs and scatter results to all the processors by using the same MPI function with the MPI_SUM parameter.

Table 1. The parallel flow logic for the transverse impedance model. The "Communication" column indicates data exchange between CPUs

N step	Actions	Communication
1	Determine the extrema of longitudinal macro-particle coordinates and construct longitudinal grids for x and y dimensions	+
2	Distribute and accumulate the macro-particle transverse dipole moments for each direction onto the longitudinal grids	-
3	Calculate FFT values of total dipole moments in the mesh	-
4	Convolute the FFT coefficients with transverse impedance values to get transverse kick at each point in the longitudinal grids	-
5	Sum all transverse kicks across all CPUs	+
6	Apply resulting transverse kick to every macro-particle	-

A thorough timing of the transverse impedance parallel implementation was not made, because there are only a few impedance elements among several hundreds of elements in a typical case and their calculation consumes very little time. There is only one requirement, namely, the single processor version and the parallel version must give the same results. We have verified that this is the case to at least six significant figures in the coordinates of macro-particles for both codes.

3 Three-Dimensional Space Charge Model

The force in our three-dimensional space charge model is calculated as the derivative of a potential, both for longitudinal and transverse components. The potential is solved as a sequence of two-dimensional transverse problems, one for each fixed longitudinal coordinate. These separate solutions are tied together in the longitudinal direction by a conducting wall boundary condition $\Phi = 0$ on the beam pipe, thus resulting in a three-dimensional potential. This method depends for its legitimacy, especially in the calculation of the longitudinal force, on the assumptions that the bunch length is much greater than the transverse beam pipe size and that the beam pipe shields out the forces from longitudinally distant particles. Although our model is applicable to long bunches, and not to the spherical bunches of interest in many linac calculations, the three-dimensional space charge model adopted here is adequate to most calculations in rings.

The three-dimensional model implemented in ORBIT closely follows a method discussed by Hockney and Eastwood [7]. A three-dimensional rectangular grid, uniform in each direction, in the two transverse dimensions and in the longitudinal coordinate is used. The actual charge distribution is approximated on the grid by distributing the particles over the grid points according to a second order algorithm, called "triangular shaped cloud (TSC)" in [7]. Then, the potential is calculated independently on each transverse grid slice, corresponding to fixed longitudinal coordinate value, as a solution of a two-dimensional Poisson's equation. The charge distribution is taken from the distribution procedure and, for the two-dimensional equation, is treated as a line charge distribution. The two-dimensional Poisson equation for the potential is then solved using fast Fourier transforms and a Green's function formulation with periodic boundary conditions [8]. The periodic boundary conditions are used only to obtain an interim solution, and this solution is then adjusted to obey the desired conducting wall boundary conditions. These are imposed on a specified circular, elliptical, or rectangular beam pipe through a least squares minimization of the difference on the beam pipe between the periodic Poisson equation solution and a superposed homogeneous solution. The homogeneous solution is represented as a series constructed from a complete set of Laplace equation solutions with variable coefficients, as described in [9]. In addition to accounting for image forces from the beam pipe, these $\Phi = 0$ boundary conditions serve to tie together the independently solved potentials from the various longitudinal slices, resulting in a self-consistent three-dimensional potential.

Finally, with the potentials determined over the three-dimensional grid, the forces on each macro-particle are obtained by differentiating the potential at the location of the macro-particle using a second order interpolation scheme. The resulting forces include both the transverse and longitudinal components. The interpolating function for the potential is the same TSC function used to distribute the charge.

The detailed description of the three-dimensional space charge algorithm can be found in [10].

4 Parallel Algorithm for the 3D Space Charge Model

The approach to parallelization of the three-dimensional space charge algorithm is obvious. We distribute the two-dimensional space charge problems for solution to different CPUs. If the number of longitudinal slices is greater than the number of CPUs, then we must group the slices. To implement this scheme it is necessary to distribute the macro-particles among the CPUs before the solving two-dimensional problems. Then, after solving the two-dimensional problems, we must provide for the exchange of neighboring transverse grids (with potentials) between CPUs to carry out the second order interpolation scheme in the longitudinal coordinate necessary for calculating and applying the space charge force kick to the macro-particles. Therefore there should be a special module that distributes macro-particles between CPUs according their longitudinal positions. We call this module "The Bunch Distributor".

4.1 The Bunch Distributor Module

The "Bunch Distributor" module analyzes the longitudinal coordinates of macro-particles currently residing on the local CPU, determines which macro-particles don't belong to this particular CPU, and sends them to the right CPU. This means that the class describing the macro-particle bunch should be a resizable container including 6D coordinates of the macro-particle and an additional flag indicating macro-particles as "alive" or "dead". This additional flag provides the possibility to have spare space in the container and to avoid changing the size of container frequently.

The logic flow for the bunch distributor module is shown in Table 2. During the two first steps we define maximum and minimum longitudinal coordinates among all macro-particles in all CPUs. To eliminate the necessity of frequent changes in the longitudinal grid we add an additional 5% to each limit and save the result. During subsequent calls of the "Bunch Distributor" module we don't change the longitudinal limits unless necessary.

After defining the longitudinal grid, we sort macro-particles according the nearest grid point. Particles that no longer belong to the appropriate CPU are stored in an intermediate buffer together with additional information about where they belong. At the step 3 we define the exchange table $N_{ex}(i,j)$ where "i" is the index of the current CPU, "j" is the index of destination CPU, and the value is the number of macro-particles that should be sent from "i" to "j". After step 4 all CPUs know the number of macro-particles they will receive. The exchange table defines the sending and receiving procedures used in step 6; therefore we avoid a deadlock. Finally, all macro-particles are located in the correct CPUs, and we can start to solve the two-dimensional space charge problems on all CPUs.

Table 2. The flow logic for the "Bunch Distributor" module. The "Communication" column indicates data exchanging between CPUs

N stage	Actions	Communication
1	Determine the extrema of longitudinal macro-particle coordinates	-
2	Find the global longitudinal limits throughout all CPUs	+
3	Analyze macro-particle longitudinal coordinates to determine on which CPU they belong. Storing the 6D macro-particle coordinates to be exchanged in an intermediate buffer and mark these macro-particles as "dead". Define an exchange table $N_{ex}(i,j)$ (see text for the explanation)	-
4	Sum the exchange table throughout all CPUs by using the MPI_Allreduce MPI function with the MPI_SUM operation parameter	+
5	Check the spare place in the bunch container and resize it if necessary	-
6	Distribute the 6D macro-particle coordinates in the intermediate buffer to the correct CPUs according the exchange table. Store the received coordinates in the bunch container in the available places	+

4.2 Parallel 3D Space Charge Algorithm

In the parallel version of the three-dimensional space charge algorithm each CPU performs the same calculation of the potential on the transverse grids as the non-parallel version. There is no need for communication between CPUs, because the macro-particles have already been distributed between CPUs by the “Bunch Distributor” module and each CPU uses its own information to solve its own segment of the longitudinal grid. There is only one difference between parallel and non-parallel versions: In the parallel version there are two additional longitudinal slices beyond the ends of the CPU’s own segment. Therefore the number of longitudinal slices for one CPU is $N_{slices}/N_{CPU}+2$ instead of N_{slices}/N_{CPU}, where N_{slices} is the total number of the transverse grid and N_{CPU} is the number of CPUs. The two additional slices are necessary because of the second order interpolation scheme. After the solution of the two-dimensional problems, the potential values from the two transverse grids on the ends of the segment should be sent to the CPU that is the neighbor according its index. In same fashion, the local CPU should obtain the potential values from its neighbors and add these potentials to its own. In this case the results of the parallel and non-parallel calculations will be the same.

5 Timing of Parallel Algorithm for the 3D Space Charge Model

Timings of the parallel algorithms were performed to elucidate the contributions of different stages in the total time of calculation and the parallel efficiency of their implementation. To avoid the effects of other jobs running on the same machine and other random factors, we did the timings on the Linux cluster with no other users and computed the average time for a number of iterations. We were able to use only five CPUs of our cluster because of the dual CPU effect.

5.1 Dual CPU Effect

Using two CPUs on the one node for parallel calculations drops the performance of our applications down by 20-30%. To clarify this situation we wrote a simple example that does not use communication between CPUs.

The executable code of the example

```
double xx[50000];
double r_arr [50000];
...
time_start = MPI_Wtime();
for( int j = 0 ; j < 275 ; j++){
 for (int i = 0; i < 50000 ; i++){
   x = x/(x+0.000001);
   r_arr[i] = x;
   xx[i] = x;
 }}
 time_stop = MPI_Wtime();
```

The execution time of the example is 1 sec for 1 CPU and 1.7 sec for 2 CPUs on the one dual CPU node. When we comment lines 07 and 08, the execution time does not depend on the number and sort of CPU's. This means that there is a competition between two CPUs with synchronized tasks on the one node for the access to the RAM if the 512 kBytes L2 cache of each CPU is not enough for data and code. To avoid this type of competition and the resulting performance drop, we use no more than 5 CPUs for each parallel run. This effect is significant for synchronized tasks only, so we can run two different parallel simulations at one time.

5.2 Timing of the Bunch Distributor Module

The timing of the bunch distributor module was carried out without including additional MPI functions in the code of the module. We measured the time needed to distribute macro-particles between CPUs according to their longitudinal positions when we have N_{part} previously distributed and N_{rand} undistributed macro-particles.

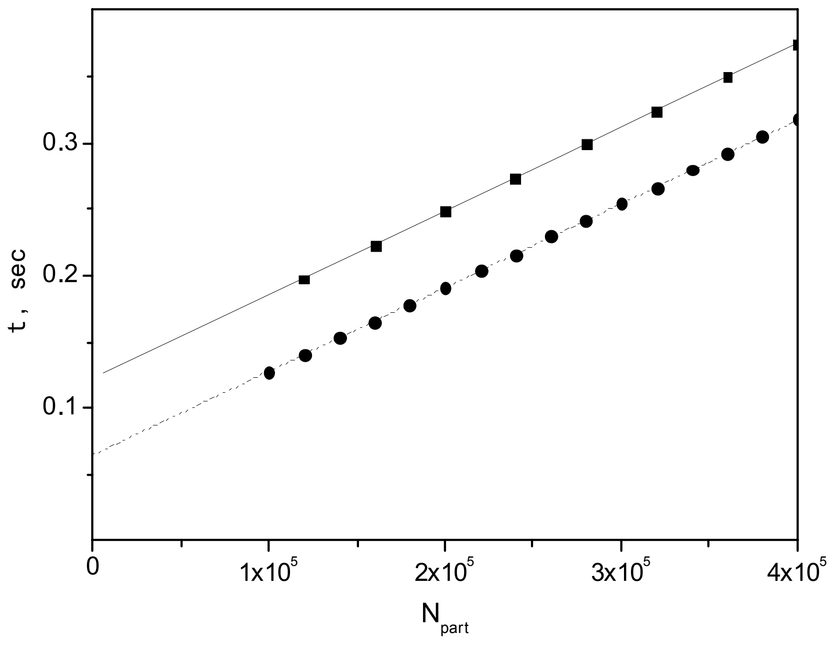

Fig. 1. The time required by the bunch distributor module to distribute N_{rand} between 2 CPUs in addition to N_{part} already distributed. The points are results of measurements, and the lines are linear approximations. The squares and circles denote N_{part} = 20000 and 10000 macro-particles accordingly.

Figure 1 shows the required time vs. N_{part} for 2 CPUs and N_{rand} = 20000 and 10000. As we expected, this time consists of two parts. The first part is proportional to the number of previously distributed particles. This is the time require for carrying out steps 1 and 3 in Table 2. The second part is proportional to the number of undistributed macro-particles that are distributed among CPUs during the step 6. Step 5 is normally carried out only once. The total execution time of steps 2 and 4 in Table 2 does not exceed 0.001 second for our case with $N_{CPU} < 6$. If the number of CPUs is large, for instance several tens, the execution time of step 4 could reduce the efficiency. In this case the parallel algorithm should be improved by using the fact that, for the long bunches found in rings, the macro-particles move very slowly along the longitudinal axis and the data exchange will be only between neighboring CPUs. This enables us to use the exchange table with 2 times N_{CPU} size instead of N_{CPU} times N_{CPU}. The analysis of graphs for several numbers of CPUs gives us the following approximation for the distribution time

$$t_{dist} = \tau_1 \cdot N_{part} / N_{CPU} + \tau_2 \cdot \alpha \cdot N_{part} \cdot (N_{CPU} - 1)/(N_{CPU} \cdot N_{CPU}) \qquad \mathbf{(1)}$$

where the parameters τ_1 and τ_2 are equal to 1.35E-6 and 12.5E-6 sec accordingly. The parameter α in Eq. (1) is the fraction of macro-particles that have to be distributed. In our simulations α is between 0 and 1E-3. Equation (1) demonstrates full scalability of this parallel algorithm.

5.3 Timing of the Parallel 3D SC Model

For timing the parallel implementation of the three-dimensional space charge model, we used a procedure analogous to that described in the previous part of this report. The calculation times were measured as a function of the number of macro-particles, number of CPUs, and 3D grid size. Fitting the measurements, we obtained the following formula for the time of calculation with the (N_x x N_y) transverse grid size and N_z longitudinal slices

$$t_{3D} = \tau_3 \cdot N_{part} / N_{CPU} + \tau_4 \cdot (N_x N_y N_z) / N_{CPU} + \tau_{comm} \cdot (N_x N_y) \qquad \mathbf{(2)}$$

where the parameters τ_3, τ_4, and τ_{comm} are 3.3E-6, 3.8E-7, and 2.1E-6 sec, respectively. The first term in the formula (2) describes the time spent binning the macro-particles, applying the space charge kick, etc. The second term is the time required to solve the set of two-dimensional space charge problems, and the last is the time for communication and is proportional to the amount of exchanged data.

Equation (2) was obtained for a uniform distribution of macro-particles along longitudinal axis. If the macro-particles are not distributed uniformly in the longitudinal direction, we should use the maximum number of macro-particles on one CPU instead of N_{part}/N_{CPU} expression in Eq. (2).

5.4 Parallel Efficiency

Using Eqs. (1) and (2) we can define the parallel efficiency of the whole algorithm as follows:

$$\eta = 100\% \cdot (t_{dist}(N_{CPU} = 1) + t_{3D}(N_{CPU} = 1))/(N_{CPU} \cdot (t_{dist} + t_{3D})) \qquad \mathbf{(3)}$$

For the cases of 64x64x64 grid, 200000 macro-particles, and 2,3,4, and 5 CPUs we obtained 98.6, 98, 97, and 96 %, respectively. These results are for a uniform distribution of the macro-particles along the longitudinal direction. If we suppose that one CPU contains 40% of all particles, instead of 20%, the parallel efficiency will be only 60%. To avoid this effect we should allocate the longitudinal slices between CPUs irregularly to provide a homogeneous load on all CPUs. This means that we must incorporate the timing results into the assignment of the longitudinal slices to the CPUs.

5 Conclusions

Parallel algorithms of the transverse impedance and the three-dimensional space charge models are developed. These algorithms provide close to 100% parallel efficiency for uniform longitudinal distributions of macro-particles. For uneven distributions of particles, the algorithms should be changed to achieve even loading and optimal performance.

6 Acknowledgments

The authors wish to thank Mike Blaskiewicz, John Galambos, Alexei Fedotov, Nikolay Malitsky, and Jie Wei for many useful discussions and suggestions during this investigation.

References

1. National Spallation Neutron Source Conceptual Design Report, Volumes 1 and 2, NSNS/CDR-2/V1, 2, (May, 1997)
2. J. Galambos, J. Holmes, D. Olsen, A. Luccio, and J. Beebe-Wang, ORBIT Users Manual, http://www.sns.gov//APGroup/Codes/Codes.htm
3. V.Danilov, J. Galambos, and J. Holmes, in Proceedings of the 2001 Particle Accelerator Conference, (Chicago, 2001)
4. J. A. Holmes, V. V. Danilov, J. D. Galambos, D. Jeon, and D. K. Olsen, Phys. Rev. Special Topics – AB 2, (1999) 114202
5. K. Woody, J. A. Holmes, V. Danilov, and J. D. Galambos, in Proceedings of the 2001 Particle Accelerator Conference, (Chicago, 2001)
6. J. A. MacLachlan, FNAL TechNote, FN-446, February (1987)
7. R. W. Hockney and J. W. Eastwood, “Computer Simulation Using Particles”, Institute of Physics Publishing (Bristol: 1988)
8. J.A. Holmes, J. D. Galambos, D. Jeon, D. K. Olsen, J. W. Cobb, M. Blaskiewicz, A. U. Luccio, and J. Beebe-Wang, Proceedings of International Computational Accelerator Physics Conference, (Monterey, CA, September 1998)
9. F. W. Jones, in Proceedings of the 2000 European Particle Accelerator Conference, (Vienna, 2000) 1381
10. J. Holmes and V.Danilov, “Beam dynamics with transverse impedances: a comparison of analytic and simulated calculations”, submitted to Phys. Rev. Special Topics – AB